Supplementary Information

1 Complexity Analysis

Complexity of the layer-wise training algorithm is $O(p \times q)$ where $p$ is the complexity of the function $GetLossP$ and $q$ is the number of times it gets evaluated by $ConjugateGradient$. The dominant term for $GetLossP$ function relates to $obj_2$. Therefore to calculate $O(p)$ it is sufficient to calculate the complexity of the linear program in equation (13). Here we use Karmarkar’s polynomial bound for LP ($O(n^{3.5}L)$) where $n$ refers to the number of variables and $L$ refers to number of input bits (Karmarkar, 1984).

The number of variables used in LP for equation (13) is equivalent to the length of vector $z$ hence:

$$\|z\| = 1 + \|t\| + \|b\| + \|\epsilon_l\| + \|\epsilon_r\| = 2m + 2d + 1 \quad \text{(1)}$$

The number of bits is equivalent to the total length of remaining matrices and vectors in equation (13):

$$\|a\| + \|l\| + \|G\| + \|y\| = (2d + 1 + 2m) + (0) + (m \times (2d + 1 + 2m)) + (m) = 2m^2 + 4m + 4d + 1 \quad \text{(2)}$$

Note that we assumed $0$ entries for $l$ given that Karmarkar’s formulation assumes a lower bound of zero when applicable hence it’s not added to calculations. Therefore:

$$O(p) = O((2m + 2d + 1)^{3.5} \times (2m^2 + 4m + 4d + 1)) \quad \text{(3)}$$

Hence the complexity of the layer-wise training for each gene is:

$$O(q \times p) = O(q \times (2m + 2d + 1)^{3.5} \times (2m^2 + 4m + 4d + 1)) \quad \text{(4)}$$

Assuming $m \gg d$ and for constant $q$ the complexity is: $O(m^{5.5})$

2 Adding Constraints

We add constraints to the objective function $obj_2$ (i.e. equation (15) in main text) to control range of GNN cell activations during training. The activation function $f_\theta(x)$ can produce extremely large values in the $[0, \infty]$ range while the observed GE values are bounded. To alleviate this, we can ensure $f_\theta(v) \leq y_u$ where $v \in \mathbb{R}_{\geq 0}^d$ is input vector representing expression of $d$ TFs and $y_u \in \mathbb{R}_{\geq 0}$ is the maximum expression level observed for current gene. Specifically, we generate input vectors $v^{(1)}, v^{(2)}, \ldots, v^{(n)}$ with values in the range of observed TF expressions (random uniform distribution with $n = 100$). Therefore, we have:

$$f_\theta(v^{(i)}) \leq y_u, \forall i = 1 \ldots n \quad \text{(5)}$$

Note that, we use symbol $v$ referring to randomly generated TF values (in contrast to symbol $x$ for observed ones). Next, we explain how above constraints can be incorporated in the objective function $obj_2$.

Considering the above and $\theta^+ = \{t^*_0, t^*, b^*, p\}$ (from equation (5) in main text), we can write:

$$f_{\theta^+}(v^{(i)}) = \frac{t^*_0 + \sum_{k=1}^d t^*_k s_k^{(i)}}{1 + \sum_{k=1}^d b^*_k h_k^{(i)}} \leq y_u, \quad s_k^{(i)} = e^{p_h v_k^{(i)}}, \forall i = 1 \ldots n \quad \text{(6)}$$

Which is equivalent to:

$$t^*_0 + \sum_{k=1}^d t^*_k s_k^{(i)} - y_u \sum_{k=1}^d b^*_k s_k^{(i)} \leq y_u, \forall i = 1 \ldots n \quad \text{(7)}$$
To convert this into matrix form, we use vector \( w^* \) (same as in equation (9) in main text) and matrices \( S \) and \( \Phi \).

\[
w^* = [t_0^*, t_1^*, t_2^*, \ldots, t_d^*, b_1^*, b_2^*, \ldots, b_d^*]^T
\]  

(8)

Matrix \( S \) consists of \( s^{(i)}_k \) values (constant for given \( p \) and \( u^{(i)} \)):

\[
S = \begin{bmatrix}
  s_1^{(1)} & s_2^{(1)} & \ldots & s_d^{(1)} \\
  s_1^{(2)} & s_2^{(2)} & \ldots & s_d^{(2)} \\
  \vdots & \vdots & \ddots & \vdots \\
  s_1^{(n)} & s_2^{(n)} & \ldots & s_d^{(n)}
\end{bmatrix}_{n \times d}
\]  

(9)

Matrix \( \Phi \) is calculated using \( S \) and \( y_u \):

\[
\Phi = \begin{bmatrix} 1 \mid S \mid (-y_u \cdot S) \end{bmatrix}_{n \times (2d+1)}
\]  

(10)

Therefore equation (7) can be represented in matrix form:

\[
\Phi.w^* \preceq y_u.1
\]  

(11)

We transform this into a form consistent with the constraints of \( obj_2 \) using a vector \( \epsilon_u \in \mathbb{R}^n_{\geq 0} \):

\[
\Phi.w^* + \epsilon_u = y_u.1, \quad \text{where } \epsilon_u \succeq 0
\]  

(12)

Including this in the constraints of \( obj_2 \) (i.e. equation (15) in main text) gives us:

\[
w^* = obj_2(C, p) = \arg \min_w 1^T \epsilon_l + 1^T \epsilon_r
\]

subject to

\[
A.w^* + \epsilon_l - \epsilon_r = y
\]

\[
\Phi.w^* + \epsilon_u = y_u.1
\]

\[
\epsilon_u \succeq 0, \epsilon_l \succeq 0, \epsilon_r \succeq 0, b \succeq 0, t \succeq 0
\]

This can be transformed into standard linear programming (LP) form below and solved using LP library:

\[
w^* = \arg \min_z a^T.z
\]

subject to

\[
G.z = y
\]

\[
z \geq l
\]  

(14)

where

\[
z = [t_0, t, b, \epsilon_l, \epsilon_r, \epsilon_u]^T, \quad a = \begin{bmatrix} 0_{d \times 1} \\ 0_{d \times 1} \\ 1_{m \times 1} \\ 1_{m \times 1} \end{bmatrix}, \quad l = \begin{bmatrix} 0_{d \times 1} \\ 0_{d \times 1} \\ 0_{m \times 1} \\ 0_{m \times 1} \end{bmatrix}
\]  

(15)

\[
G = \begin{bmatrix}
  A_{m \times (2d+1)} & J_{m \times m} & -J_{m \times m} & 0_{m \times n}
  \\
  \Phi_{n \times (2d+1)} & 0_{n \times m} & 0_{n \times m} & I_{n \times n}
\end{bmatrix}
\]  

(16)
Figure 1: Our **synthetic data generation pipeline** consists of four main steps as visualized. First, the simulation experiments (single knockouts and perturbation of master regulator GE) for the given network are defined. Second, the transcription network is initialized and thermodynamic simulation executed providing GE in steady state for given experiments. Third, GEs are normalized between 0 and 1 for each gene. Fourth, for stratified sampling, hierarchical clustering is executed and one profile is sampled from each cluster.
Figure 2: Architecture illustration of the competing ANNs used. The input consists of master regulator expression for gene $a$ and the knockout vector $KO$. The output consists of gene expression for rest of the genes. (A) Example of multi-layer perceptron (MLP) architecture with five hidden nodes and sigmoid activation function. (B) Example of recurrent neural network (RNN) architecture with sigmoid activation function. Note that input for each RNN node (in red) includes the network input vector as well as activation signals from other RNN nodes. (C) Example of bi-directional Recurrent Neural Network (BiRNN) architecture with sigmoid activation function used in both directions. For each pair of sigmoids in BiRNN, the average of activations ($\bar{x}$) is used to calculate final output. In cases where expression levels are not normalized to $[0, 1]$ range (as for in-vivo data of DREAM challenge (Marbach et al., 2012)), predicted gene expressions are mapped into valid range using linear transformation. That is $(y_{\text{max}} - y_{\text{min}}) \times \hat{y} + y_{\text{min}}$, where $\hat{y}$ is predicted expression (in range of $[0, 1]$ due to sigmoid activation). The $y_{\text{min}}$ and $y_{\text{max}}$ are constant for a given gene and correspond to the minimum and maximum expressions observed for each gene in the whole dataset. To implement, we used ANN modules from Keras (Chollet et al., 2015) and TensorFlow (Abadi et al., 2015) python libraries.
Figure 3: 5-fold cross validation performance evaluation (training) using in vivo microarray data. For methods with hyper-parameters, the hyper-parameter combination with best test error is selected. A, B, C show performance of all methods on 110 randomly selected network modules on inferred transcription network of *E. coli*. GNN shows better overall performance.
Figure 4: GNN (predicted vs. observed) GE for 10 experiments in 5-Fold cross-validation setting for chemotaxis. The genes used as master regulator are cRP, cpxR, flhDC, hNS, iHF, lrhA, matA, nsrR, rbsR, rcsAB and sutR. Results show GNN’s power in predicting GE for new experiments given that 10 observed GE profiles are distinct from each other.
Figure 5: The prediction performance of 4 methods are visualized given 500 observed GE values randomly selected (no replacement) on synthetic dataset for chemotaxis. Based on $R^2$ metric, GNN performs best.
References


